

# Estructuras de Datos y Algoritmos

## Árboles AVL

### LECCIÓN 14

© All wrongs reversed – bajo licencia CC-BY-NC-SA 4.0



**Universidad**  
Zaragoza

Dpto. de Informática e Ingeniería de Sistemas  
Universidad de Zaragoza, España

Curso 2024/2025

**Grado en Ingeniería Informática**  
UNIVERSIDAD DE ZARAGOZA

*Aula 0.04, Edificio Agustín de Betancourt*



Adaptadas de diapositivas de Javier Campos

# Índice

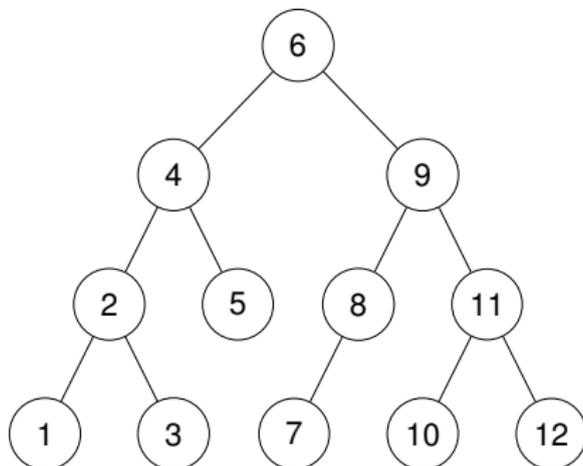
- 1 Conceptos de árboles binarios equilibrados
- 2 Fundamentos de árboles AVL
- 3 Implementación de árboles AVL
  - Inserción
  - Borrado
- 4 Implementación de árboles AVL (pseudocódigo)

# Índice

- 1** Conceptos de árboles binarios equilibrados
- 2 Fundamentos de árboles AVL
- 3 Implementación de árboles AVL
- 4 Implementación de árboles AVL (pseudocódigo)

# Conceptos

- Un árbol binario de búsqueda se dice **equilibrado** (o **balanceado**) si y sólo si, para cada uno de sus nodos ocurre que las alturas de sus 2 subárboles difieren como mucho en 1
  - Definición dada por Adelson-Velskii y Landis (1962)
  - Denominados **árboles AVL**



## Árboles balanceados o equilibrados

- Un ABB es *k-equilibrado* si cada nodo lo es
- Un nodo es *k-equilibrado* si las alturas de sus subárboles izquierdo y derecho difieren en no más de  $k$  ( $k = 0$  sería equilibrio perfecto)

## Árboles AVL

- Definidos por Adelson-Velskii y Landis (1962)
- Un ABB 1-equilibrado se llama AVL

# Índice

- 1 Conceptos de árboles binarios equilibrados
- 2 Fundamentos de árboles AVL**
- 3 Implementación de árboles AVL
- 4 Implementación de árboles AVL (pseudocódigo)

# Fundamentos de árboles AVL

## En un árbol AVL:

- Se garantiza que la altura del árbol es mínima, o casi:  $\Theta(\log N)$ , siendo  $N$  el número de nodos del árbol
  - Teorema de Adelson-Velskii y Landis

# Fundamentos de árboles AVL

## En un árbol AVL:

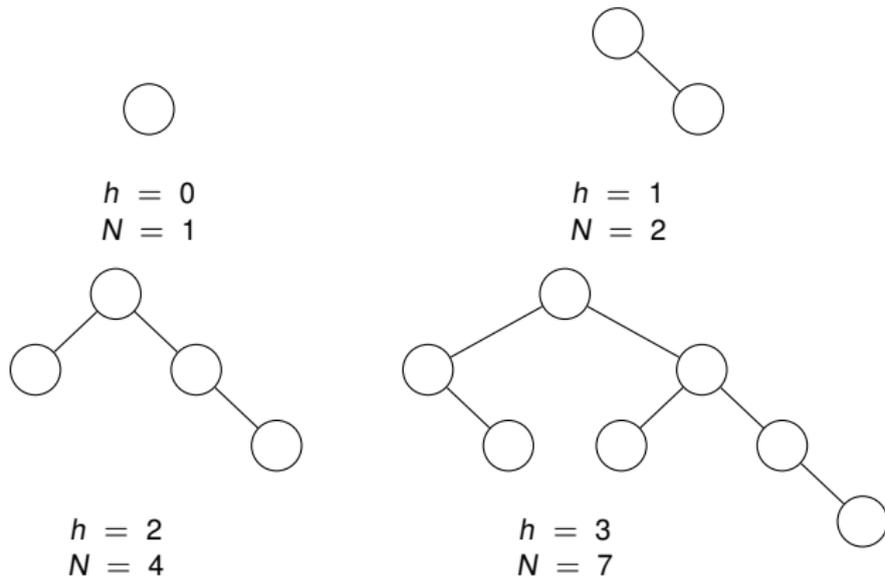
- Se garantiza que la altura del árbol es mínima, o casi:  $\Theta(\log N)$ , siendo  $N$  el número de nodos del árbol
  - Teorema de Adelson-Velskii y Landis
- Coste de inserción, búsqueda y borrado, en el peor caso:  $O(\log N)$

# Teorema de Adelson-Velskii y Landis

*¿Cuál es el mínimo número de nodos en un AVL de altura  $h$ ?*

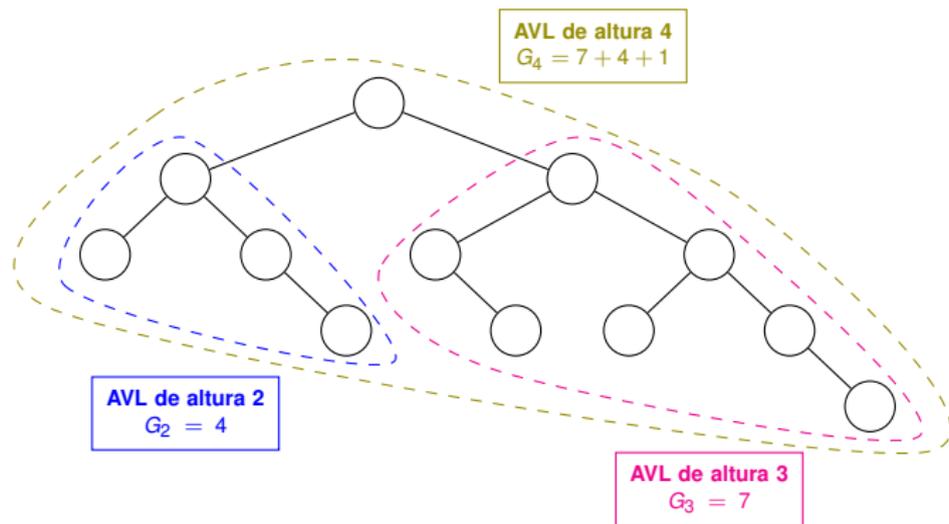
# Teorema de Adelson-Velskii y Landis

¿Cuál es el mínimo número de nodos en un AVL de altura  $h$ ?



# Teorema de Adelson-Velskii y Landis

- Sea  $G_h$  el número de nodos en el AVL de altura  $h$  que tiene el mínimo número posible de nodos
- Entonces:  $G_h = G_{h-1} + G_{h-2} + 1$



# Teorema de Adelson-Velskii y Landis

- Por su semejanza y relación con la serie de Fibonacci:

$$F_h = F_{h-1} + F_{h-2}; F_0 = 1; F_{-1} = 0$$

- A los árboles AVL construidos con el menor número de nodos posible para una altura dada se les denomina *árboles de Fibonacci*

$h$	$G_h$	$F_h$
0	1	1
1	2	1
2	4	2
3	7	3
4	12	5
5	20	8
6	33	13
...	...	...

- Como se tiene que  $G_h = F_{h+2} - 1$

- Y se sabe que  $F_h > \frac{\phi^h}{\sqrt{5}} - 1$ , con  $\phi = \frac{(1 + \sqrt{5})}{2}$

- Entonces:  $G_h > \frac{\phi^{h+2}}{\sqrt{5}} - 2$

# Teorema de Adelson-Velskii y Landis

## Teorema

- La altura  $h$  de un árbol AVL con  $N$  nodos internos siempre está comprendida entre  $\log_2(N + 1)$  y  $1.4404 \log_2(N + 2) - 0.3277$ .

## Demostración

- Un AVL de altura  $h$  tiene como máximo  $2^h$  nodos. Entonces:

$$N + 1 \leq 2^h \Rightarrow h \geq \log_2(N + 1)$$

- Para encontrar el límite superior, plantearemos el problema de encontrar el número mínimo de nodos internos contenidos en un árbol equilibrado de altura  $h$

- Sea  $T_h$  un AVL de altura  $h$  con el mínimo número de nodos posible. Por definición de AVL, un subárbol de  $T_h$  tendrá una altura de  $h - 1$ , y el otro tendrá  $h - 1$  o  $h - 2$ . Como  $T_h$  tiene el mínimo número de nodos posible, vamos a considerar que el subárbol izquierdo tiene una altura  $h - 1$  y el subárbol derecho  $h - 2$ .
- Sea  $T_{h-1}$  el subárbol izquierdo y  $T_{h-2}$  el subárbol derecho. Entonces, se demuestra por inducción que el árbol de Fibonacci de altura  $h + 1$  tiene los menos nodos posibles de entre todos los posibles árboles balanceados de altura  $h$ . Entonces:

$$N \geq F_{h+2} - 1 > \frac{\phi^{h+2}}{\sqrt{5}} - 2; \sqrt{5}(N + 2) > \phi^{h+2}; \frac{1}{2} \log_2 5 + \log_2(N + 2) > (h + 2) \log_2 \phi \Rightarrow$$

$$\Rightarrow h < 1.4404 \log_2(N + 2) - 0.3277$$

# Índice

- 1 Conceptos de árboles binarios equilibrados
- 2 Fundamentos de árboles AVL
- 3 Implementación de árboles AVL**
  - Inserción
  - Borrado
- 4 Implementación de árboles AVL (pseudocódigo)

# Implementación de árboles AVL

- Definición de **factor de equilibrio**:  $F_e = h_{\text{subárbol derecho}} - h_{\text{subárbol izquierdo}}$
- Permite construir algoritmos más simples que considerando las alturas
- Si el factor de equilibrio de un nodo es:
  - 0: el nodo está **perfectamente equilibrado** (es decir, sus subárboles tienen exactamente la misma altura)
  - 1: el nodo está **pesado a derechas** (es decir, su subárbol derecho es un nivel más alto)
  - 1: el nodo está **pesado a izquierdas** (es decir, su subárbol izquierdo es un nivel más alto)
- Hay que recalcular el factor de equilibrio al insertar o borrar elementos
- Tras insertar o borrar, si  $F_e \geq 2$  o  $F_e \leq -2$ , es necesario **reequilibrar**

# Implementación de árboles AVL

- Hay que guardar información sobre el equilibrio en cada nodo

```
tipos avl = ↑nodo
  factorEquil = (pesadoIzq, equilibrado, pesadoDer);
                { o lo que es lo mismo, -1, 0, 1 }
nodo = registro
      laClave: clave; { o bien, dato: elemento }
      elValor: valor;
      equilibrio: factorEquil;
      izq, der: avl
freg
```

# Implementación de árboles AVL

## Proceso de inserción

### Pasos

- 1 Buscar hasta encontrar la posición de inserción o modificación (proceso idéntico a inserción en árbol binario de búsqueda → se insertará siempre en una hoja)
- 2 Insertar el nuevo nodo, con factor de equilibrio “equilibrado”
- 3 Desandar el camino de búsqueda, verificando el equilibrio de los nodos del camino, y reequilibrándolos, si es necesario

# Implementación de árboles AVL

## Proceso de inserción

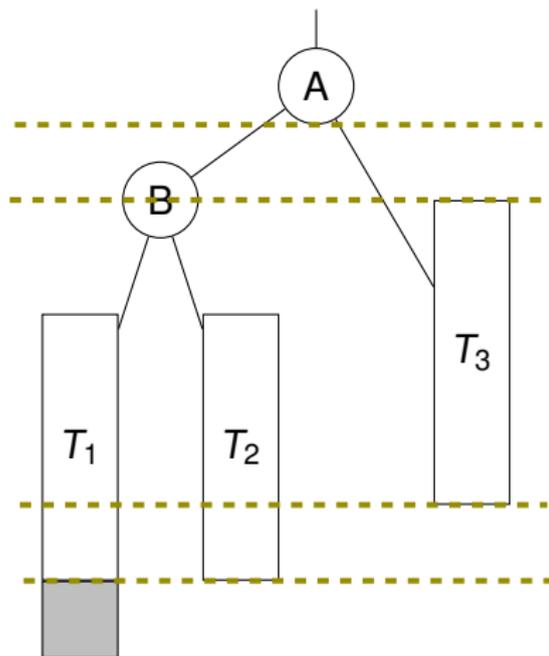
### ■ Se implementa mediante recursividad

- Se pasa una variable que **indica si el subárbol ha aumentado de tamaño**
- Si no ha aumentado, no cambia el factor de equilibrio
- Si ha aumentado, hay varias posibilidades:

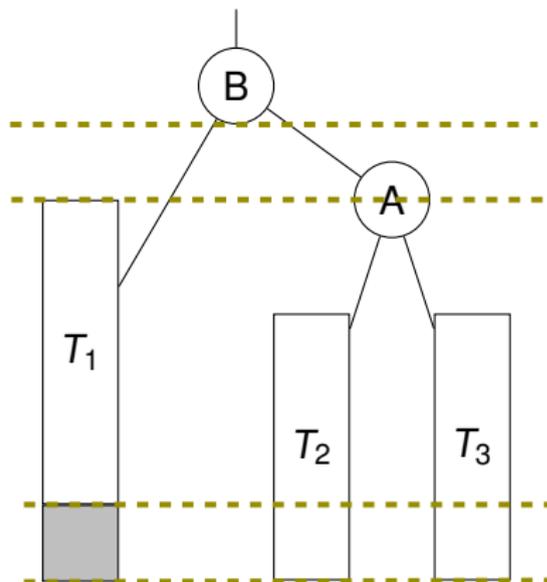
Caso	Insertado en la izquierda	Insertado en la derecha
Era pesadoDer	Ahora equilibrado	<b><i>Hay que reequilibrar</i></b>
Era equilibrado	Ahora pesadoIzq	Ahora pesadoDer
Era pesadoIzq	<b><i>Hay que reequilibrar</i></b>	Ahora equilibrado

# Implementación de árboles AVL

## Proceso de inserción – posibles casos



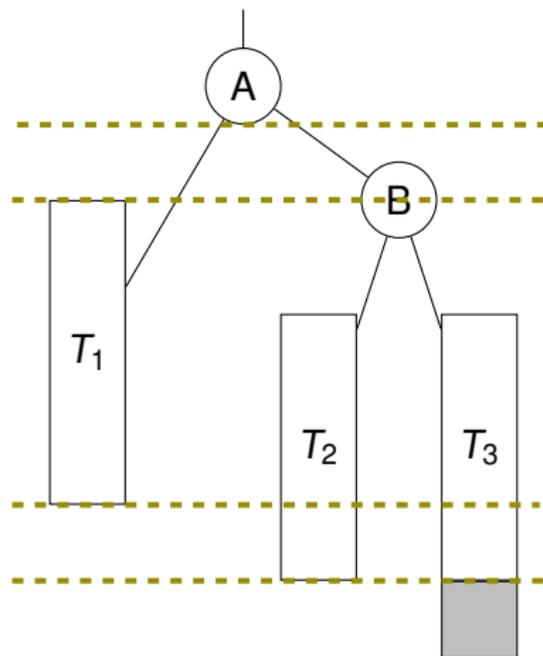
**Caso 1: Izquierda-Izquierda**



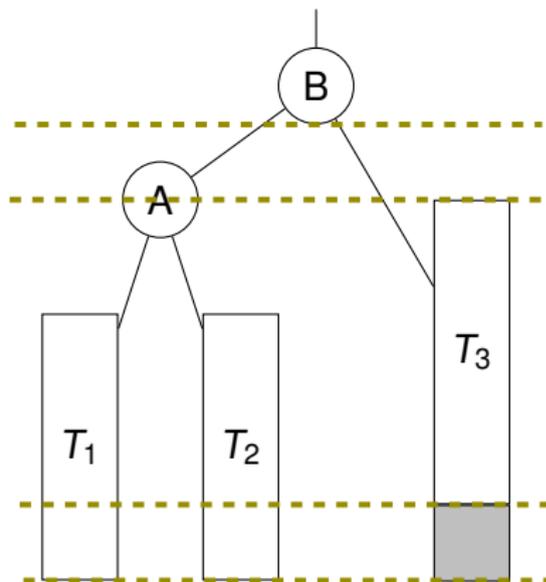
**Solución rotación a la izquierda**

# Implementación de árboles AVL

## Proceso de inserción – posibles casos



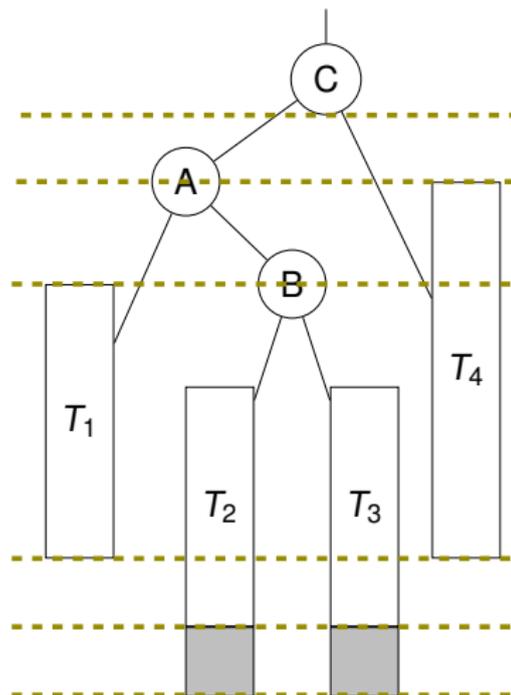
**Caso 2: Derecha-Derecha**



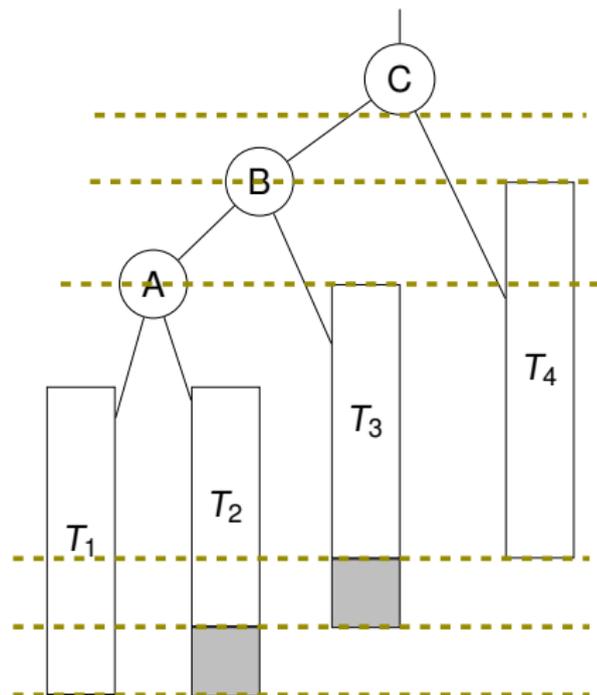
**Solución** *rotación a la derecha*

# Implementación de árboles AVL

## Proceso de inserción – posibles casos



**Caso 3: Izquierda-Derecha**

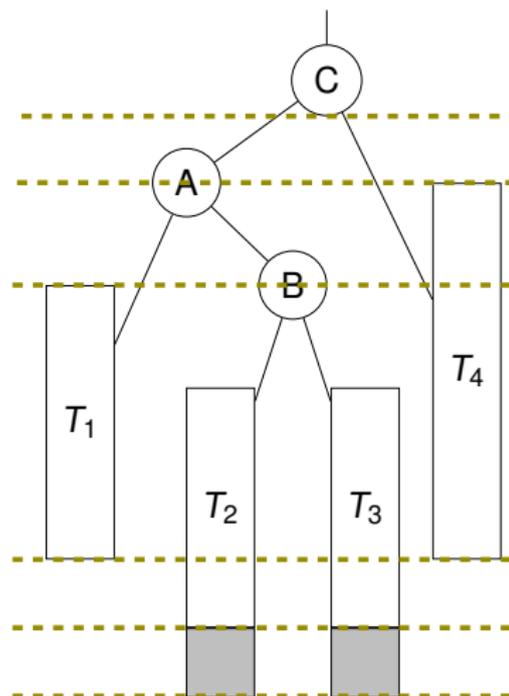


**Solución rotación doble**

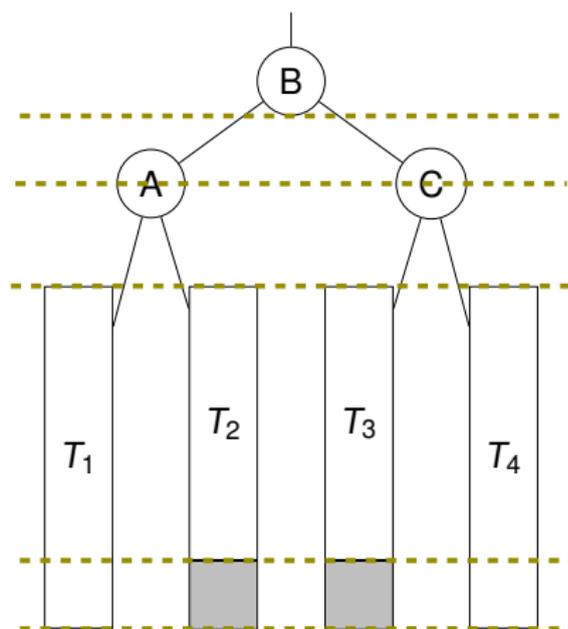
**PASO 1: rotación a derecha**

# Implementación de árboles AVL

## Proceso de inserción – posibles casos



**Caso 3: Izquierda-Derecha**

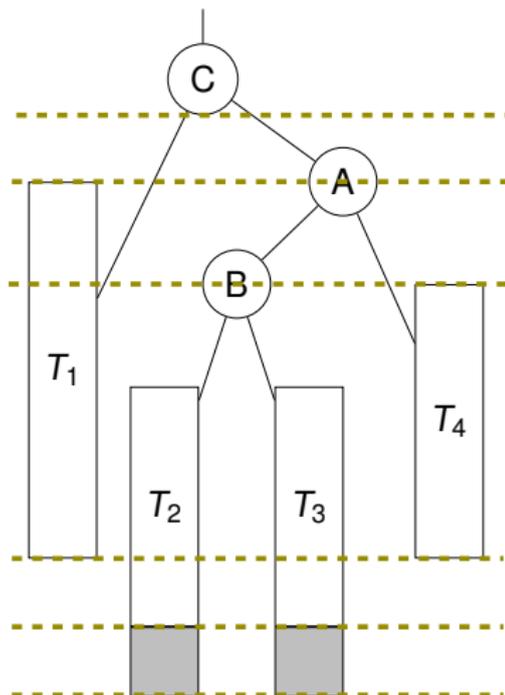


**Solución** *rotación doble*

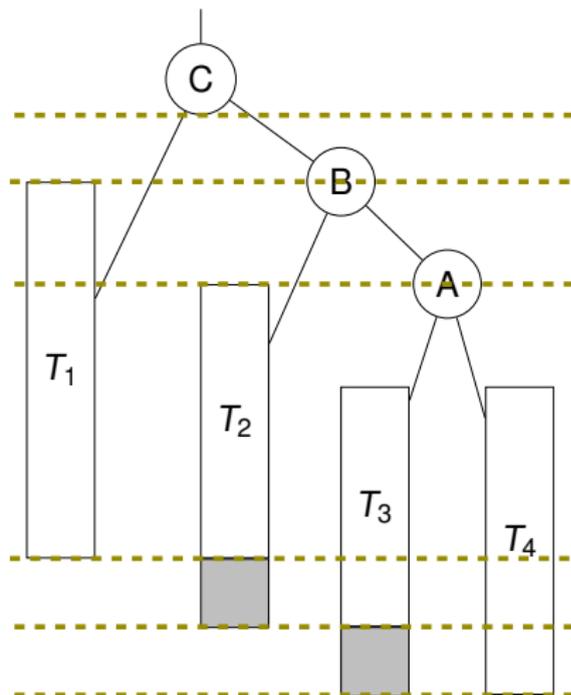
**PASO 2: rotación a izquierda**

# Implementación de árboles AVL

## Proceso de inserción – posibles casos de desequilibrios



**Caso 4: Derecha-Izquierda**

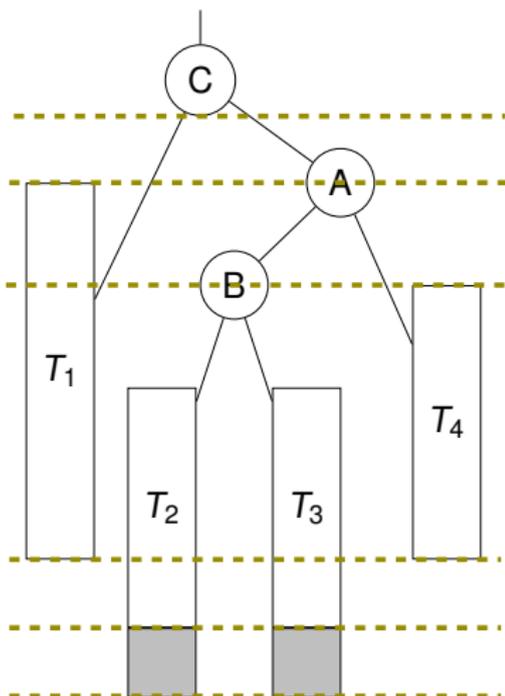


**Solución rotación doble**

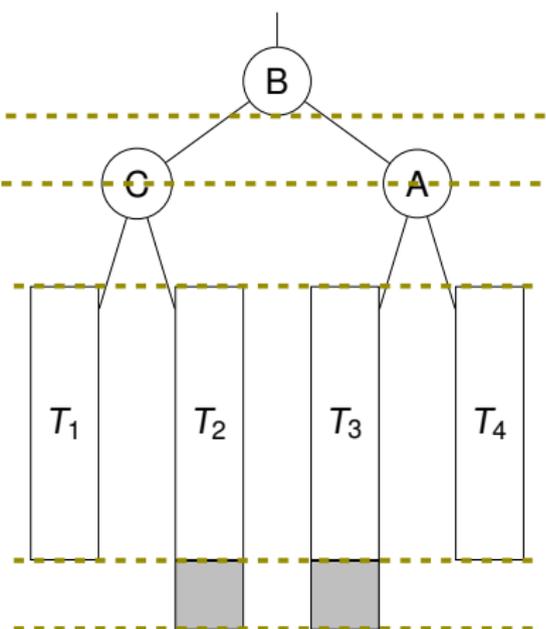
Paso 1: *rotación a izquierda*

# Implementación de árboles AVL

## Proceso de inserción – posibles casos de desequilibrios



**Caso 4: Derecha-izquierda**



**Solución** *rotación doble*

PASO 2: *rotación a derecha*  Universidad Zaragoza

# Implementación de árboles AVL

## Proceso de inserción – notas finales

### Importante

- El nodo insertado se establece como equilibrado
- Se regresa por el camino de búsqueda, recalculando el factor de equilibrio de los nodos, hasta alcanzar la raíz o encontrar un nodo que no cumpla el factor de equilibrio, y requiera una reestructuración para reequilibrar
- Tras reequilibrar, el árbol resultante queda de la misma altura que en su estado inicial
  - Por lo tanto, si inicialmente estaba equilibrado, el árbol resultante tras la inserción, también estará equilibrado
- **Basta una única rotación** (u operación de reequilibrar) para dejar todo el árbol resultante equilibrado
  - Es decir, no hace falta seguir evaluando hasta la raíz

# Implementación de árboles AVL

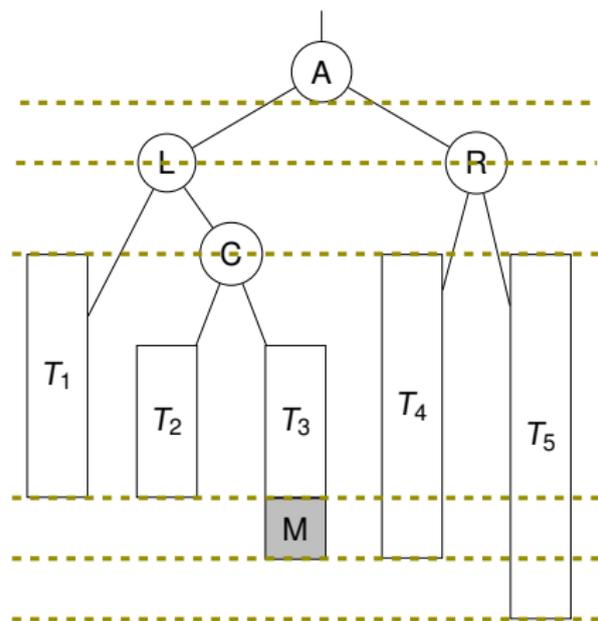
## Proceso de borrado

### **Pasos** (*similar al borrado en ABB*)

- 1 Localizar el nodo a borrar
- 2 Se borra el nodo, atendiendo a:
  - Si el nodo es hoja, se borra
  - Si no es hoja:
    - Se sustituye por el máximo del subárbol izquierdo; y
    - Se borra dicho máximo del subárbol izquierdo
- 3 Se regresa por el camino de búsqueda, calculando los nuevos factores de equilibrio:
  - Si algún nodo pierde la condición de equilibrio, debe ser restaurada → **reequilibrar**
  - **Debe continuarse hasta la raíz, porque pueden ser necesarios más reequilibrados**

# Implementación de árboles AVL

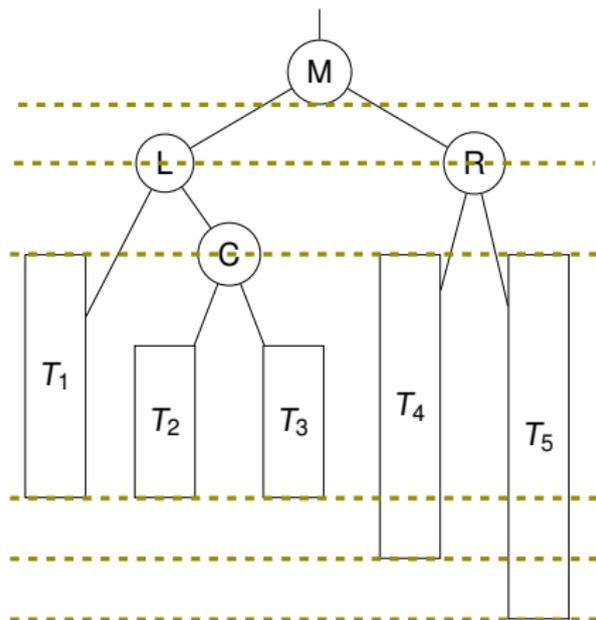
## Proceso de borrado – ejemplo



- **Borrar A y la nueva raíz será M**

# Implementación de árboles AVL

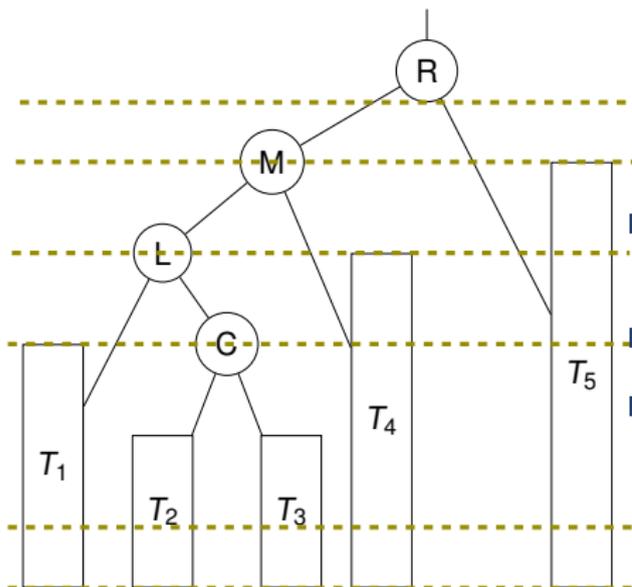
## Proceso de borrado – ejemplo



- Borrado A, la nueva raíz es M. **Árbol resultante pesado a la derecha**
- Solución similar a la inserción: aplicar **rotación a la derecha**

# Implementación de árboles AVL

## Proceso de borrado – ejemplo



- Borrado A, la nueva raíz es M. **Árbol resultante pesado a la derecha**
- Aplicada rotación a la **derecha**
- **El árbol resultante ha perdido altura**

En borrado pueden ser necesarias varias operaciones de restauración del equilibrio, y hay que seguir comprobando hasta llegar a la raíz

# Índice

- 1 Conceptos de árboles binarios equilibrados
- 2 Fundamentos de árboles AVL
- 3 Implementación de árboles AVL
- 4 Implementación de árboles AVL (pseudocódigo)**

# Implementación de árboles AVL (pseudocódigo)

módulo árbolesAVL

exporta

tipo avl

procedimiento vacío(sal a: avl);

procedimiento insertar(e/s a: avl; ent c: clave; ent v: valor);

procedimiento borrar(e/s a: avl; ent c: clave);

procedimiento buscar(ent a: avl; ent c: clave; sal éxito: booleano; sa

implementación

tipos avl = ↑nodo

factorEquil = (pesadoIzq, equilibrado, pesadoDer);  
{ o lo que es lo mismo, -1, 0, 1 }

nodo = registro

laClave: clave; { o bien, dato: elemento }

elValor: valor;

equilibrio: factorEquil;

izq, der: abb

freg

```

procedimiento insertar(e/s a: avl; ent c: clave; ent v: valor)
variable
    alturaModificada: booleano
principio
    alturaModificada := falso;
    insertarRec(a, c, v, alturaModificada)
fin

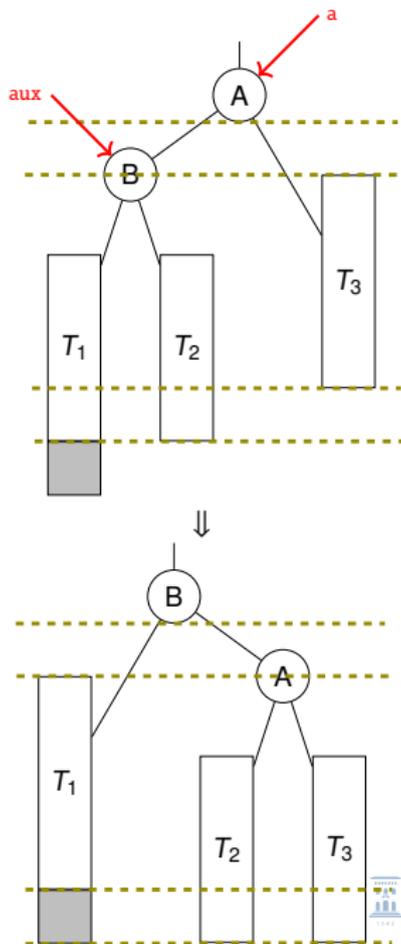
procedimiento insertarRec(e/s a: avl; ent c: clave; ent v: valor; e/s alturaModificada: booleano)
principio
    si a = nil entonces
        nuevodato(a);
        a↑.laClave := c; a↑.elValor := v;
        a↑.equilibrio := equilibrado;
        a↑.izq := nil; a↑.der := nil;
        alturaModificada := verdad; {se ha modificado su altura}
    sino_si c < a↑.laClave entonces
        insertarRec(a↑.izq, c, v, alturaModificada);
        si alturaModificada entonces
            selección
                a↑.equilibrio=pesadoIzq:
                    si a↑.izq↑.equilibrio = pesadoIzq entonces
                        rotaciónIzq(a)
                    sino
                        rotaciónIzqDer(a)
                    fsi;
                    alturaModificada := falso;
                a↑.equilibrio = equilibrado: a↑.equilibrio := pesadoIzq;
                a↑.equilibrio = pesadoDer: a↑.equilibrio := equilibrado;
                alturaModificada:=falso
            fselección
            fsi
        sino_si a↑.laClave < c entonces
            {... simétrico ...}
        sino
            a↑.elValor := v {actualizar el valor asociado a la clave}
        fsi
    fin

```

```

procedimiento rotaciónIzq(e/s a :avl)
variable
  aux: avl
principio
  aux := a↑.izq;
  a↑.izq := aux↑.der;
  a↑.equilibrio := equilibrado;
  aux↑.der := a;
  a := aux;
  a↑.equilibrio := equilibrado
fin

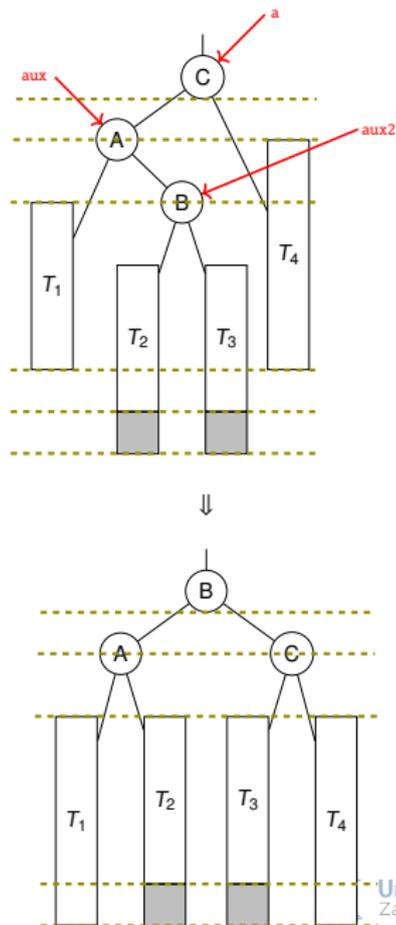
```



```

procedimiento rotacionIzqDer(e/s a: avl)
variables
    aux1, aux2: avl
principio
    aux1 := a↑.izq;
    aux2 := a↑.izq.↑der;
    aux1↑.der := aux2↑.izq;
    aux2↑.izq := aux1;
    a↑.izq := aux2;
si aux2↑.equilibrio = pesadoIzq entonces
    aux1↑.equilibrio := equilibrado;
    a↑.equilibrio := pesadoDer
sino_si aux2↑.equilibrio = equilibrado entonces
    aux1↑.equilibrio := equilibrado;
    a↑.equilibrio := equilibrado
sino
    aux1↑.equilibrio := pesadoIzq;
    a↑.equilibrio := equilibrado
fsi;
    a↑.izq := aux2↑.der;
    aux2↑.der := a;
    aux2↑.equilibrio := equilibrado;
    a := aux2
fin

```



```

procedimiento borrar(e/s a: avl; ent c: clave)
variable
    alturaModificada: booleano
principio
    alturaModificada := falso;
    borrarRec(a, clave, alturaModificada)
fin

procedimiento borrarRec(e/s a: avl; ent c: clave; e/s alturaModificada: booleano)
variable
    aux: avl
principio
    si a ≠ nil entonces
        si c < a↑.laClave entonces
            borrarRec(a↑.izq, c, alturaModificada);
            si alturaModificada entonces
                equilIzq(a, alturaModificada)
            fsi
        sino_si a↑.laClave < c entonces
            borrarRec(a↑.der, c, alturaModificada);
            si alturaModificada entonces
                equilDer(a, alturaModificada);
            fsi
        sino
            si a↑.izq = nil entonces
                aux := a;
                a := a↑.der;
                disponer(aux);
                alturaModificada := verdad
            sino_si a↑.der = nil entonces
                aux := a;
                a := a↑.izq;
                disponer(aux);
                alturaModificada := verdad
            sino
                borrarMaxClave(a↑.izq, a↑.laClave, a↑.elValor, alturaModificada);
                si alturaModificada entonces
                    equilIzq(a, alturaModificada);
                fsi
            fsi
        fsi
    fsi
fin

```

```

procedimiento equilIzq(e/s a: avl; e/s alturaModificada: booleano)
procedimiento equilDer(e/s a: avl; e/s alturaModificada: booleano)
{... estudio de casos (similar inserción)...}

procedimiento borrarMaxClave(e/s a: avl; sal c: clave; sal v: valor; e/s alturaModificada: booleano)
variable
  aux: avl
principio
  si a↑.der = nil entonces
    c := a↑.laClave;
    v := a↑.elValor;
    aux := a;
    a := a↑.izq;
    disponer(aux);
    alturaModificada := verdad
  sino
    borrarMaxClave(a↑.der, c, v, alturaModificada);
    si alturaModificada entonces
      equilDer(a, alturaModificada);
    fsi
  fsi
fin

procedimiento buscar(ent a: avl; ent c: clave; sal éxito: booleano; sal v: valor)
{... igual que en árboles binarios de búsqueda...}

```

## Applets

- <http://people.ksp.sk/~kuko/bak/>
- <http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html>
- <http://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

# Estructuras de Datos y Algoritmos

## Árboles AVL

### LECCIÓN 14

© All wrongs reversed – bajo licencia CC-BY-NC-SA 4.0



**Universidad**  
Zaragoza

Dpto. de Informática e Ingeniería de Sistemas  
Universidad de Zaragoza, España

Curso 2024/2025

**Grado en Ingeniería Informática**  
UNIVERSIDAD DE ZARAGOZA

*Aula 0.04, Edificio Agustín de Betancourt*

